The Big Bro - LogiTrace

Patrones de diseño aplicados (Oct 2025)

1. Singleton para la conexión a la base de datos

Contexto:

Django expone el ORM, pero los servicios de dominio necesitaban ejecutar SQL de diagnóstico sin duplicar conexiones ni credenciales.

Solución:

`infraestructura/database.py` define `DatabaseConnection`, un *Singleton thread-safe* que comparte `connections['default']` en todo el *stack*.

Implementación:

`__new__` controla la *instancia única* y `_wrapper()` inicializa una sola vez el *handler* de Django bajo demanda. `execute()` devuelve filas como diccionarios para reutilizarlas sin *boilerplate*.

Uso:

`domain/services/database_health.py` consume el *Singleton* para medir latencia y `core/views.database_health` expone `/health/db/` en `logitrace/urls.py`.

Beneficios:

(1) un único punto de configuración, (2) menos conexiones abiertas, (3) facilidad para instrumentar métricas y auditoría del *driver*.

Riesgos:

si la conexión cae, `DatabaseConnection` cierra y reintenta en la siguiente solicitud. Recomendado monitorear el *endpoint* desde DevOps.

Extensiones:

el *Singleton* puede exponer *helpers* para transacciones largas o inyectarse en futuros repositorios de dominio.

2. Strategy para políticas de *picking* (propuesto)

Escenario:

LogiTrace debe alternar entre **FIFO**, **FEFO** o **prioridad por cliente** según la orden y la zona de almacén.

Propuesta:

cada algoritmo implementa la interfaz `PickingStrategy` con el método `seleccionar_items(orden)`. El servicio `OrderFulfillment` elige la estrategia adecuada en tiempo de ejecución.

Beneficios esperados:

nuevas reglas sin tocar el resto del código, pruebas unitarias específicas por estrategia y configuración por bodega.

Implementación futura:

`domain/strategies/picking.py` con clases **FIFO**, **Expiración** y **Prioridad**, inyectadas en el servicio de cumplimiento.

Con esto se cubre el requisito obligatorio (*Singleton* operativo) y se deja documentado el escenario para un segundo patrón extensible.